Design and Implementation of a focused crawler using Association based classification

A Research Proposal

For the Partial fulfillment of Degree of

Doctor of Philosophy

In

Computer Engineering

By

Smita Vijayvargiya

Under the Guidance of

Dr. Sonal Jain

Associate Professor

Department of Computer Engineering,

JK LAXMIPAT UNIVERSITY

Research Proposal

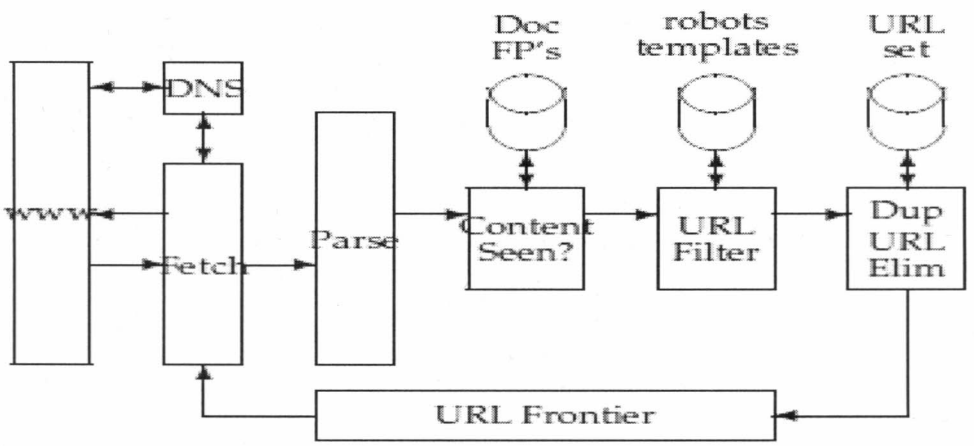
Topic: Design and Implementation of a focused crawler using association rule based classification

Introduction

The heart of search engine is web crawler. A Web crawler (also called Web robot or Web spider) is a program that automatically traverses the Web's hypertext structure by retrieving a document, and recursively retrieving all documents that are referenced. Web crawlers are often used as resource discovery and retrieval tools for Web search engines. The quality of a crawler directly affects the searching quality of such web search engines. Such a web crawler may interact with millions of hosts over a period of weeks or months

The basic operation of any hypertext crawler is as follows. The crawler begins with one or more URLs that constitute a *seed set*. It picks a URL from this seed set, and then fetches the web page at that URL. The fetched page is then parsed, to extract both the text and the links from the page (each of which points to another URL). The extracted text is fed to a text indexer. The extracted links (URLs) are then added to a *URLfrontier*, which at all times consists of URLs whose corresponding pages have yet to be fetched by the crawler. Initially, the URL frontier contains the seed set; as pages are fetched, the corresponding URLs are deleted from the URL frontier. The entire process may be viewed as traversing the web graph In continuous crawling, the URL of a fetched page is added back to the frontier for fetching again in the future. This seemingly simple recursive traversal of the web graph is complicated by the many demands on a practical web crawling system: the crawler has to be distributed, scalable, efficient, polite, robust and extensible while fetching pages of high quality. Fetching a billion pages (a small fraction of the static Web at present) in a month-long crawl requires fetching several

hundred pages each second. A multi-threaded design is needed to address several bottlenecks in the overall crawler system in order to attain this fetch rate.



Because the Web is gigantic and being constantly updated, the design of a good crawler poses many challenges The world-wide web, having over 146.31.5 million pages continues to grow rapidly (according to a Nature magazine article1, the World Wide Web doubles in size approximately every 8 months). Such growth poses basic limits of scale for today's generic crawlers and search engines. In last years of 90th decade, Alta Vista's crawler, called the Scooter, was running on a 1.5GB memory, 30GB RAID disk, 4x533MHz AlphaServer 4100 5/300 with 1 GB/s I/O bandwidth. In spite of these heroic efforts with high-end multiprocessors and clever crawling software, the largest crawls cover only 30-40% of the web, and refreshes take weeks to a month 3.. Therefore it is essential to develop effective crawling strategies to prioritize the pages to be indexed.

Challenges in implementing a crawler

Given this size and change rate of the Web, the crawler needs to address many important challenges, including the following:

1. **What pages should the crawler download?** In most cases, the crawler cannot download *all* pages on the Web. Even the most comprehensive search engine currently indexes a small fraction of the entire Web [6]. Given this fact, it is important for the crawler to carefully select the pages and to visit "important" pages first, so that the fraction of the Web that is visited (and kept up-to-date) is more meaningful.

2. **How should the crawler refresh pages?** Once the crawler has downloaded a significant number of pages, it has to start *revisiting* the downloaded pages in order to detect changes and refresh the downloaded collection. Because Web pages are changing at very different rates [7], the crawler needs to carefully decide which pages to revisit and which pages to skip in order to achieve high "freshness" of pages. For example, if a certain page rarely changes, the crawler may want to revisit the page less often, in order to visit more frequently changing ones.

3. **How should the load on the visited Web sites be minimized?** When the crawler collects pages from the Web, it consumes resources belonging to other organizations [Kos95]. For example, when the crawler downloads page p on site S, the site needs to retrieve page p from its file system, consuming disk and CPU resources. After this retrieval the page then needs to be transferred through the network, which is another resource shared by multiple organizations. Therefore, the crawler should minimize its impact on these resources. Otherwise, the administrators of a Web site or a particular network may complain and sometimes may completely block access by the crawler.

4. **How should the crawling process be parallelized?** Due to the enormous size

of the Web, crawlers often run on multiple machines and download pages in parallel. This parallelization is often necessary in order to download a

In particular, the crawler must deal with huge volumes of data. Unless it has unlimited computing resources and unlimited time, it must carefully decide what URLs to download and in what order. We here discuss an important challenge: How should a crawler select URLs to download from its list of known URLs? If a crawler intends to perform a single scan of the entire Web, and the load placed on target sites is not an issue, then any URL order will suffice. That is, eventually every single known URL will be visited, so the order is not critical. However most crawlers will not be able to visit every possible page for two main reasons:

· The crawler or its client may have limited storage capacity, and may be unable to index or analyze all pages. Currently the Web is believed to have several terabytes of textual data and is growing rapidly, so it is reasonable to expect that most clients will not want or will not be able to cope with all that data

· Crawling takes time, so at some point the crawler may need to start revisiting previously retrieved pages, to check for changes. This means that it may never get to some pages. It is currently estimated that there exist more than one billion pages available on the Web and many of these pages change at rapid rates In either case, it is important for the crawler to visit “important” pages first, so that the fraction of the Web that is visited (and kept up to date) is more meaningful

Focused Crawlers

Focused crawlers are programs designed to selectively retrieve Web pages relevant to a specific domain for the use of domain specific search engines and digital libraries. Unlike the simple crawlers behind most general search engines which collect any reachable Web pages in breadth-first order, focused crawlers try to "predict" whether or not a target URL is pointing to a relevant and high-quality Web page before actually fetching the page. In addition, focused crawlers visit URLs in an optimal order such that URLs pointing to relevant and high-quality Web pages are visited first, and URLs that point to low-quality or irrelevant pages are never visited. There has been much research on algorithms designed to determine the quality of Web pages. However, most focused crawlers use local search algorithms such as *best- first search* to determine the order in which the target URLs are visited.

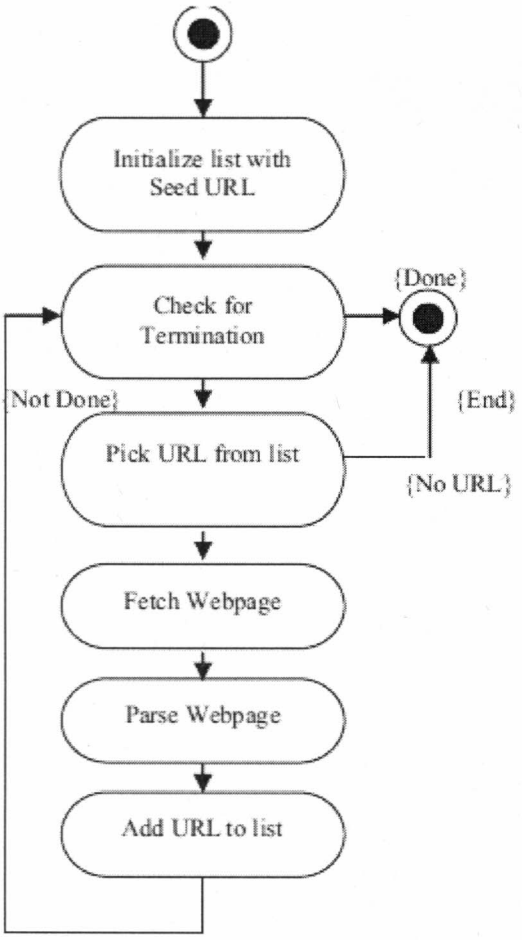


Figure 1. Generic Crawler sequence

*Lit*erature review

Web crawlers have been studied since the advent of the Web [1,2]. Since many crawlers can download only a small subset of the Web, crawlers need to carefully decide which pages to download in [3] can discover and identify "important" pages early, and propose some algorithms to achieve this goal.

Early algorithms

Starting with the early breadth first, exhaustive crawlers (Pinkerton, 1994) and depth first crawlers such as Fish Search Web crawling was simulated by a "group of fish" migrating on the web. In the so called "fish search" each URL corresponds to a fish whose survivability is dependent on visited page relevance and remote server speed. Page relevance is estimated using a binary classification (the page can only be relevant or irrelevant) by a means of a simple keyword or regular expression match. Only when fish traverse a specified amount of irrelevant pages they die off - that way information that is not directly available in one 'hop' can still be found. On every document the fish produce offspring – its number being dependant on page relevance and the number of extracted links. The school of fish consequently 'migrates' in the general direction of relevant pages which are then p resented as results. Starting point is specified by the user by providing 'seed' pages that are used to gather initial URLs.

URLs are added to the beginning of the crawl list which makes this a sort of a depth first search.[Hersovici98] extends this algorithm into "shark-search". URLs of pages to be downloaded are prioritized by taking into account a linear combination of source page relevance, anchor text and neighborhood (of a predefined size) of the link on the source page and inherited relevance score. Inherited relevance score is parent page's relevance score multiplied by a specified decay factor. Unlike in [DeBra94] page relevance is calculated as a similarity between document and query in vector space model and can be any real number between 0 and 1. Anchor text and anchor context scores are also calculated as similarity to the query.

[Cho98] propose calculating the PageRank [Page98] score on the graph induced by pages downloaded so far and then using this score as a priority of URLs extracted from a page. They show some improvement over the standard breadth-first algorithm. The improvement however is not large. This may be due to the fact that the PageRank score is calculated on a very small, non-random subset of the web and also that the PageRank algorithm is too general for use in topic-driven tasks.

3. Crawling with the help of background knowledge

[Chakrabarti99] use existing document taxonomy (e.g pages in Yahoo tree) and seed documents to build a model for classification of retrieved pages into categories (corresponding to nodes in the taxonomy). The use of taxonomy also helps at better modeling of the negative class: irrelevant pages are usually not drawn from a homogenous class but could be classified in a large number of categories with each having different properties and features. In this paper the same applies for the positive class because the user is allowed to have interest in several non-related topics at the same time. The system is built from 3 separate components: crawler, classifier and distiller. The classifier is used to determine page relevance (according to the taxonomy) which also determines future link expansion. Two different rules for link expansion are presented. Hard focus rule allows expansion of links only if the class to which the source page belongs with the highest probability is in the 'interesting' subset. Soft focus rule uses the sum of probabilities that the page belongs to one of the relevant classes to decide visit priority for children; no page is eliminated a priori. Periodically the distiller subsystem identifies hub pages (using a modified hubs & authorities algorithm [Kleinberg98]). Top hubs are then marked for revisiting Experiments show almost constant average relevance of 0.3 – 0.5 (averaged over 1000 URLs). Quality of results retrieved using unfocused crawler almost immediately drops to practically 0.In [Chakrabarti02] page relevance and URL visit priorities are decided by separate models. The model for evaluating page relevance can be anything that outputs a binary classification, but the model for URL ranking (also called "apprentice") is on-line trained by samples consisting of source page features and the relevance of the target page (that kind of information is of course available only after both the source and the target page have been downloaded and the target page evaluated for relevance). For each retrieved page, the apprentice is trained on information from baseline (in this case the aforementioned taxonomy model) classifier (i.e. with what probability does the parent page belong to some class) and features around the link extracted from the parent page - to predict the relevance of the page pointed to by the link. Those predictions are then used to order URLs in the crawl priority queue. Number of false positives is shown to

decrease significantly – between 30% and 90%. [Ehrig03] consider an ontology-based algorithm for page relevance computation. After preprocessing, entities (words occurring in the ontology) are extracted from the page and counted. Relevance of the page with regard to user selected entities of interest is then computed by using several measures on ontology graph (e.g. direct match, taxonomic and more complex relationships). The harvest rate is improved compared to the baseline focused crawler (that decides on page relevance by a simple binary keyword match) but is not compared to other types focused crawlers.[Bergmark02] describe modified 'tunneling' enhancement to best-first focused crawler approach. Since relevant information can sometimes be located only by visiting some irrelevant pages first and since the goal is not always to minimize the number of downloaded pages but to Soumen Chakrabarti, Martin Van den Berg, and Byron Dom were the first to propose a soft-focus crawler, which obtains a given page's relevance score (relevance to the target topic) from a classifier and assigns this score to every URL extracted from that page. We refer to this soft-focus crawler as the baseline focused crawler.

Focused crawling with tunneling

An essential weakness of the baseline focused crawler is its inability to model tunneling; that is, it can't tunnel toward the on-topic pages by following a path of off-topic pages.5 Two remarkable projects, the context-graph-based crawler and Cora's focused crawler, achieve tunneling.The context-graph-based crawler uses a best-search heuristic,but the classifiers learn the layers representing a set of pages that are at some distance to the pages in the target class (layer 0).6 The crawler simply uses these classifier results and inserts URLs extracted from a layer-*i* page to the layer-*i* queue; that is, it keeps a edicated queue for each layer. While deciding the next page to visit, the crawler prefers the pages nearest to the target class—that is, the URLs popped from the queue that correspond to the first nonempty layer with the smallest layer label. This approach clearly solves the problem of tunneling, but unfortunately requires constructing the context graph,

which in turn requires finding pages with links to a particular page (back links). Rule-based crawler, on the other hand, uses forward links while generating the rules and transitively combines these rules to effectively imitate tunneling behavior.

Focused Crawler by Jun Lee en al [10] used anchor text tag and decision tree to design . The number of terms in an instance of anchor text is small compared to that of the whole content of a web page. To effectively exploit the information contained in anchor texts, a decision tree is employed to predict the relevance of the target pages.

Reinforcement learning RL is employed[11] by Ioannis Partalas1, Georgios Paliouras2, Ioannis Vlahavas to develop focused crawler for selecting an appropriate classifier that in turn evaluates the links that the crawler must follow. The introduction of link classifiers reduces the size of the search space for the RL method and makes the problem tractable.

Proposed Approach

Association rule mining (association rule learning) – Association rule learning is employed to discover interesting relations between variables in large databases. This dependency modeling analyses strong rules discovered in databases using different measures of interestingness.

Classification – Classification is the procedure of assigning labels to objects such that objects' labels within the same categories will match previously labeled objects from a training set, by generalizing known structure. Classification is traditionally a type of supervised learning problem that tries to learn a function from the data in order to predict the categorical labels unknown objects to differentiate between objects of different lasses. Classification procedure can be employed to assist decision makers to classify

alternatives into multiple groups, reduce the number of misclassifications and lessen the impact of outliers

Particularly, our work focused on the approach of association rule mining, which extracts knowledge from data sets and the knowledge discovered is represented by rules. At a very abstract level,knowledge can be repre sented by links between items, whereas items are facts or events. These links of items will be referred as rules. These rules can permit a system to order and organize its interaction with its environment, giving the possibilities for reasoning such as predicting events, and other analyses. Agrawal et al. first presented the concept of strong rules, where association rules are used to discover regularities between products (modeled by sets of items) in large-scale databases (Agrawal, Imielinski, & Swami, 1993).

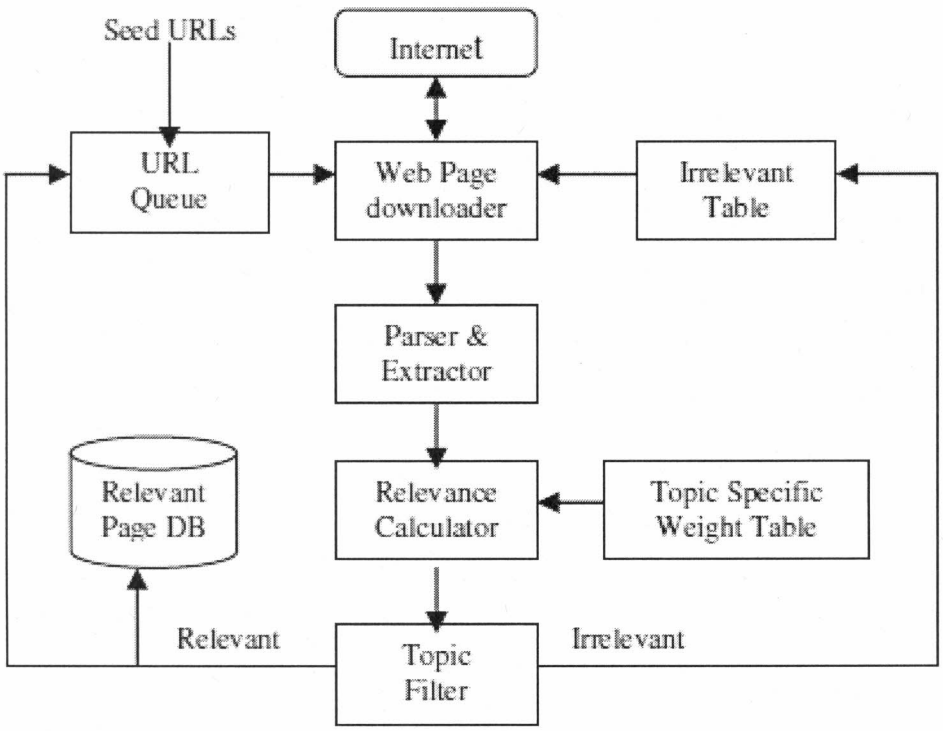
More formally, the problem of association rule mining is stated as follows (Agrawal et al., 1993).

Let I = {a1,a2, . . .,an} be a finite set of items. A transaction database is a set of ransactions T = {t1, t2, . . ., tm} where each transaction tj # I (1 6 j 6 m) represents a set of items. An itemset is a set of items X # I. The support of an itemset X is denoted as sup(X) and is defined as the number of transactions that contain X. An association rule X?Y is a relationship between two itemsets X,Y such that X, Y # I and X \ Y = Ø. The support of a rule X?Y is defined as sup(X?Y) = sup(X[Y)/|T|. The confidence of a rule X?Y is defined as conf(X?Y) = sup(X[Y)/sup(X). The problem of mining association rules is to find all association rules in a database having a support no less than a user-defined threshold minsup and a confidence no less than a user-defined threshold minconf. The problem of rule mining can be decomposed in two steps: Step 1is to determine all frequent itemsets in the database (itemsets being present in at least minsup _ |T| transactions). Step 2 is to discover association rules by using the frequent itemsets found n step

1. For each frequent itemset X, pairs of frequent itemsets P and Q = X _ P are carefully chosen to engender rules of the form P?Q. For each such rule P?Q, if sup(P?Q)Pminsup andconf(P

?Q)Pminconf, the rule is output.A subset of the problem of association rule mining is the problem of mining sequential rules common to several sequences as follows. A sequence database SD is a set of sequences S = {s1, s2, . . .,sn} and a set of items I = {i1, i2, . . ., in}, where each sequence sx is an ordered list of itemsets sx = {X1,X2,. . .,Xn} such that X1,X2, . . .,Xn # I. An item x is said to occur before another item y in a sequence sx = {X1,X2, . . .,Xn} if there exists integers k < m such that x 2 Xk and y 2 Xm. A sequential rule X)Y is defined as a relationship between two itemsets X, Y # I such that X \ Y = Ø and X, Y are not empty. The interpretation of a sequential rule X)Y is that if the items of X occur in some itemsets of a sequence, the items in Y will occur in some itemsets afterward in the same sequence. The problem of mining sequential rules common to several sequences is to find all sequential rules from a sequence database such that their support and confidence are respectively higher or equal to some user-defined thresholds minSup and minConf. More generally, frequent patterns are itemsets, subsequences, or substructures that appear in a data set with frequency no less than a user-specified threshold. A substructure can refer to various structural forms, such as subgraphs, subtrees, or sublattices, which may be combined with itemsets or ubsequences (Han, Cheng, Xin,& Yan, 2007). Frequent pattern mining plays an essential role in association rule mining. For instance, the design knowledge concerning a given task can be specified through frequent pattern mining used to search for frequently occurring design diagrams that are represented as attributed hierarchical layout hypergraphs encoding knowledge engaged for reasoning about design features.

Proposed Model



References

[1] Oliver A. McBryan. GENVL and WWWW: Tools for taming the web. In *Proceedings of the First World-Wide Web Conference*, Geneva, Switzerland, May 1994.

[2] Brian Pinkerton. Finding what people want: Experiences with the web crawler. In *Proceedings of the Second World-Wide Web Conference*, Chicago, Illinois, October 1994.

[3] Junghoo Cho, Hector Garcia-Molina, and Lawrence Page. Efficient crawling through URL ordering. In *Proceedings of the Seventh International World-Wide Web Conference*, Brisbane, Australia, April 1998.

[4] P. M. E. De Bra and R. D. J. Post, "Information retrieval in the World-Wide Web: Making client-based searching feasible," *Computer Networks and ISDN Systems*, vol. 27, no. 2, pp. 183–192, 1994. [Online]. Available: citeseer.ist.psu.edu/debra94information.html

[5] M. Chau and H. Chen, "Personalized and Focused Web Spiders," *Web Intelligence*, N.Zhong, J. Liu, and Y. Yao, eds., Springer-Verlag, 2003, pp. 197–217.

[6] F. Menczer, "ARACHNID: Adaptive retrieval agents choosing heuristic neighborhoods for information discovery," in *Machine Learning: Proceedings of the Fourteenth International Conference*, 1997, pp. 227–235.

[7] Soumen Chakrabarti, Martin van den Berg, and Byron Dom. Focused crawling:

A new approach to topic-specific web resource discovery. In *Proceedings* 166 *BIBLIOGRAPHYof the Eighth International World-Wide Web Conference*, Toronto, Canada,May 1999.

[8] Krishna Bharat and Andrei Broder. Mirror, mirror on the web: A study of host pairs with replicated content. In *Proceedings of the Eighth International World-Wide Web Conference*, Toronto, Canada, May 1999.

[9] Junghoo Cho and Hector Garcia-Molina. The evolution of the web and implications for an incremental crawler. In *Proceedings of the Twenty-sixth International Conference on Very Large Databases*, Cairo, Egypt, September 2000.